US011392399B2

(12) United States Patent
Kirmse et al.

(10) Patent No.: US 11,392,399 B2
(45) Date of Patent: Jul. 19, 2022

(54) **EXTERNAL BINARY SIDECAR FOR CLOUD CONTAINERS**

(71) Applicant: **SAP SE**, Walldorf (DE)

(72) Inventors: **Daniel Kirmse**, Berlin (DE); **Henning Zahn**, Muehlenbeck (DE)

(73) Assignee: **SAP SE**, Walldorf (DE)

( * ) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 252 days.

(21) Appl. No.: **15/931,519**

(22) Filed: **May 13, 2020**

(65) **Prior Publication Data**

US 2021/0357238 A1 Nov. 18, 2021

(51) **Int. Cl.**

***G06F 9/455*** (2018.01)
***G06F 11/14*** (2006.01)

(52) **U.S. Cl.**

CPC ...... ***G06F 9/45558*** (2013.01); ***G06F 11/1464*** (2013.01); *G06F 2009/45583* (2013.01); *G06F 2009/45595* (2013.01); *G06F 2201/815* (2013.01)

(58) **Field of Classification Search**

None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

10,528,367 B1 * 1/2020 Liu ........................ G06F 9/4498
10,558,457 B1 * 2/2020 Kolb ........................ G06F 8/65
10,776,099 B2 * 9/2020 Duvur ........................ G06F 8/71
10,783,007 B2 * 9/2020 Fildebrandt ...... G06Q 10/06312
11,003,434 B2 * 5/2021 Duvur ..................... G06F 9/455
11,023,270 B2 * 6/2021 Mahajan ............. G06F 16/9027
11,068,245 B2 * 7/2021 Jaeger ................... G06F 8/4441
11,176,245 B2 * 11/2021 Nunez Mencias .... G06F 21/602
11,226,847 B2 * 1/2022 Alluboyina ........... G06F 9/4881

* cited by examiner

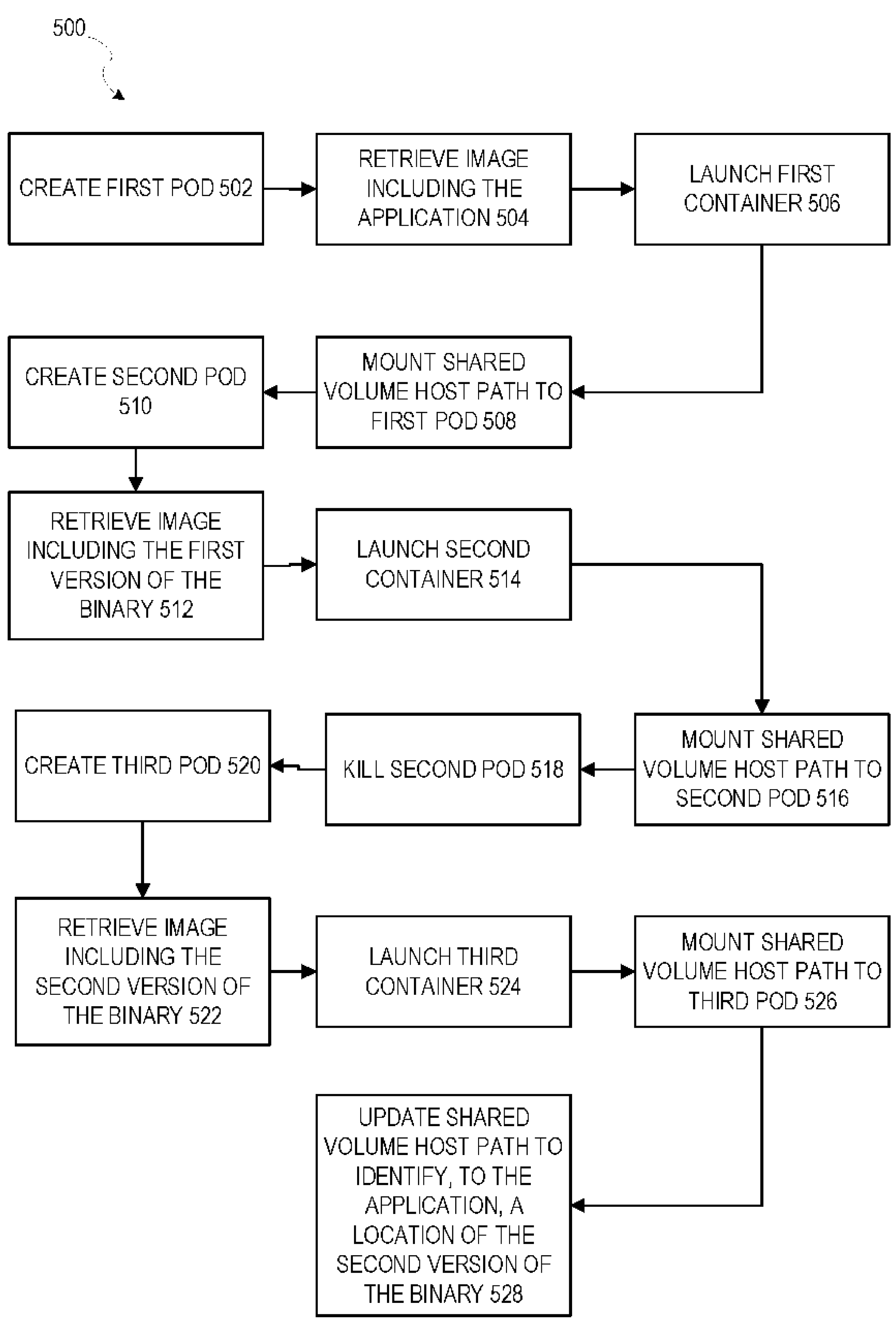
*Primary Examiner* — Van H Nguyen

(74) *Attorney, Agent, or Firm* — Schwegman Lundberg & Woessner, P.A.

(57) **ABSTRACT**

In an example embodiment, a solution is provided that causes a binary used by an in-memory database application to be deployed in a separate container. The container with the binary may be called a “sidecar” to the container with the in-memory database application. Both containers mount the same path in the hosts shared filesystem to expose the binary to the application. There is no permanent connection between the two containers, and thus the binary can be updated individually without inducing a downtime of the in-memory database application.

**20 Claims, 7 Drawing Sheets**



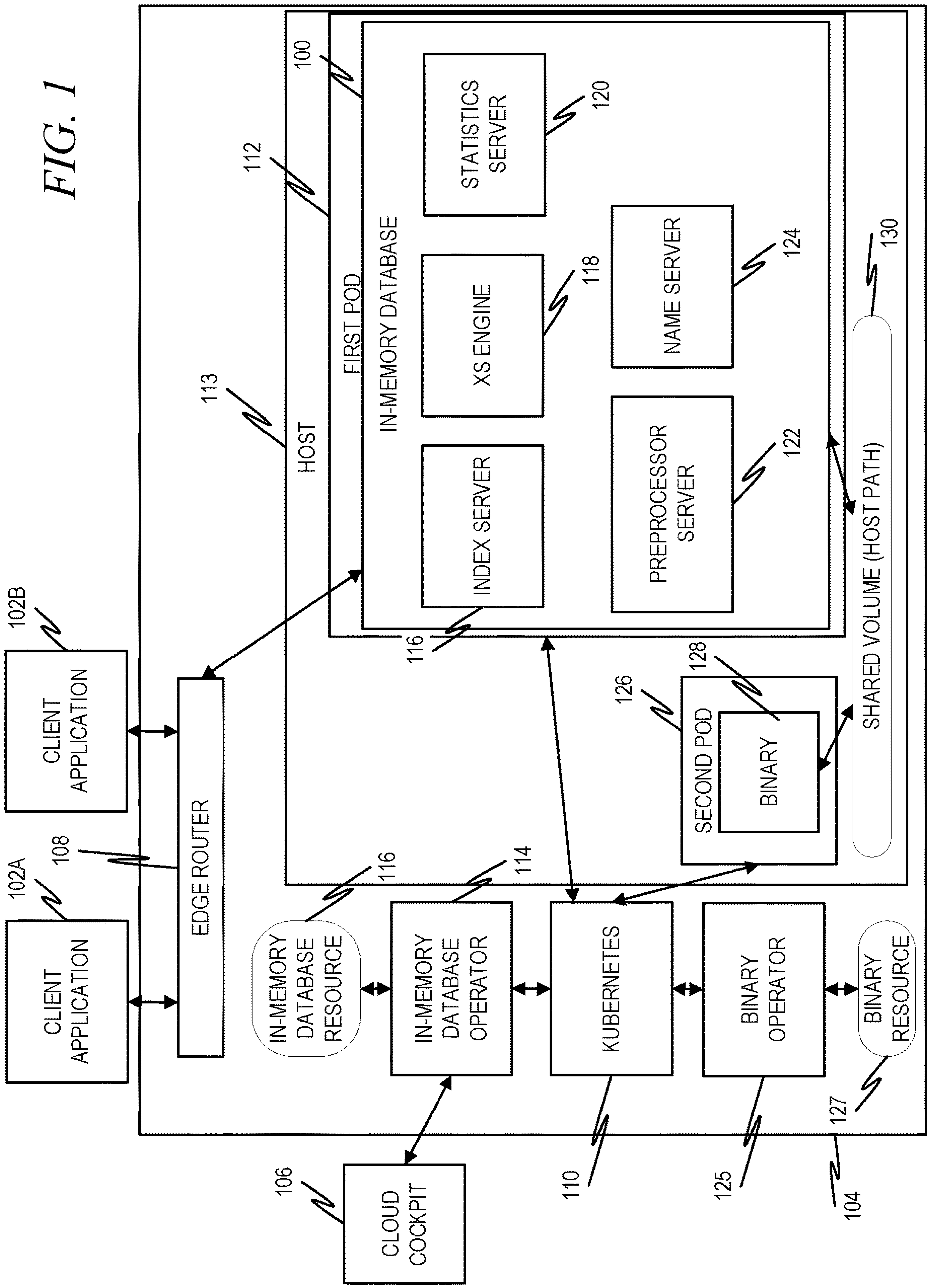

CLIENT APPLICATION
102A
CLIENT APPLICATION
102B
108
EDGE ROUTER
113
HOST
112
FIRST POD
100
IN-MEMORY DATABASE
STATISTICS SERVER
120
XS ENGINE
118
INDEX SERVER
116
NAME SERVER
124
PREPROCESSOR SERVER
122
130
SHARED VOLUME (HOST PATH)
126
SECOND POD
128
BINARY
116
IN-MEMORY DATABASE RESOURCE
114
IN-MEMORY DATABASE OPERATOR
KUBERNETES
110
BINARY OPERATOR
125
BINARY RESOURCE
127
104
CLOUD COCKPIT
106

FIG. 1

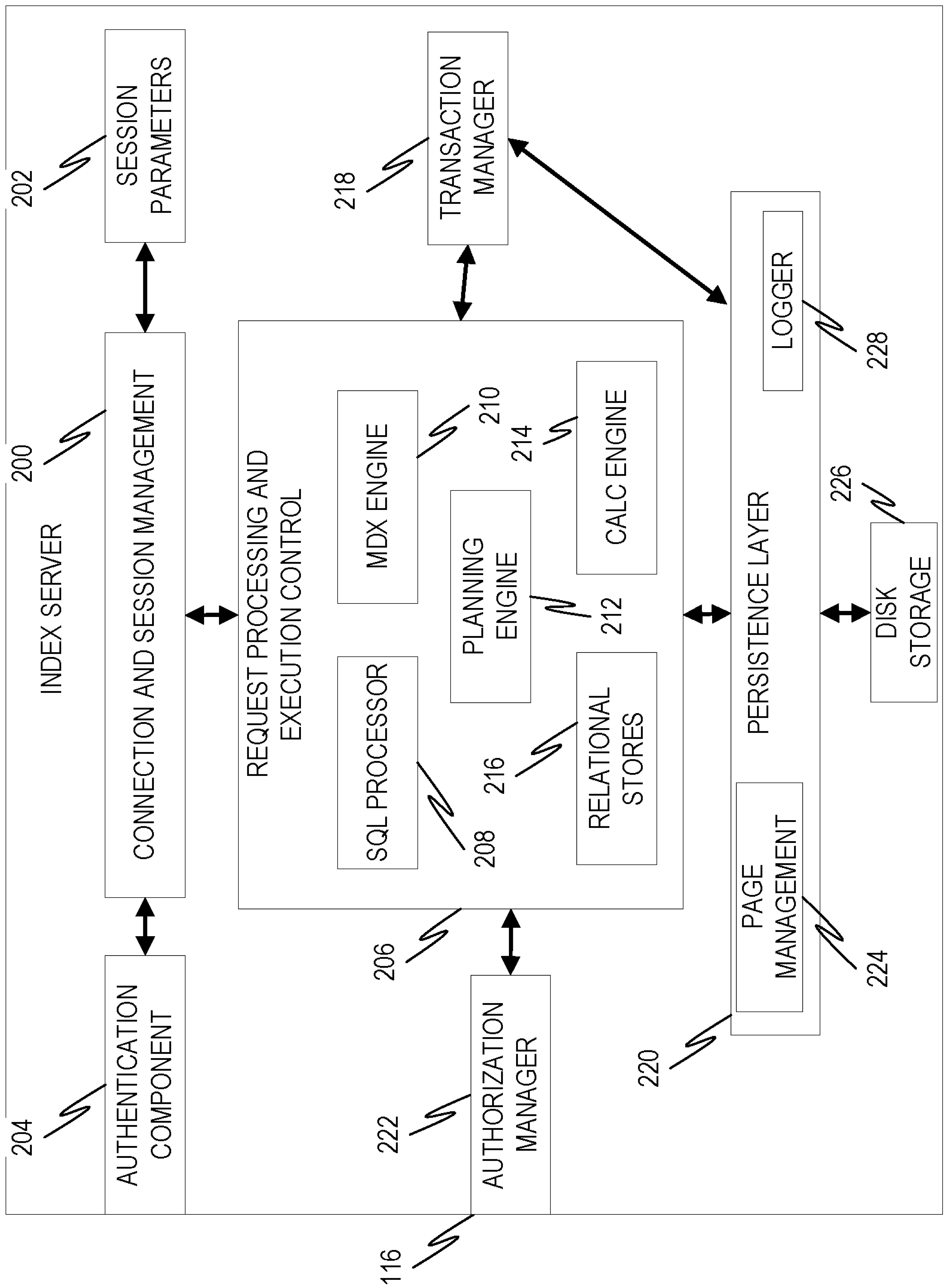

INDEX SERVER
200
CONNECTION AND SESSION MANAGEMENT
202
SESSION PARAMETERS
204
AUTHENTICATION COMPONENT
REQUEST PROCESSING AND EXECUTION CONTROL
206
SQL PROCESSOR
208
MDX ENGINE
210
PLANNING ENGINE
212
CALC ENGINE
214
RELATIONAL STORES
216
TRANSACTION MANAGER
218
PERSISTENCE LAYER
220
AUTHORIZATION MANAGER
222
PAGE MANAGEMENT
224
DISK STORAGE
226
LOGGER
228
116

FIG. 2

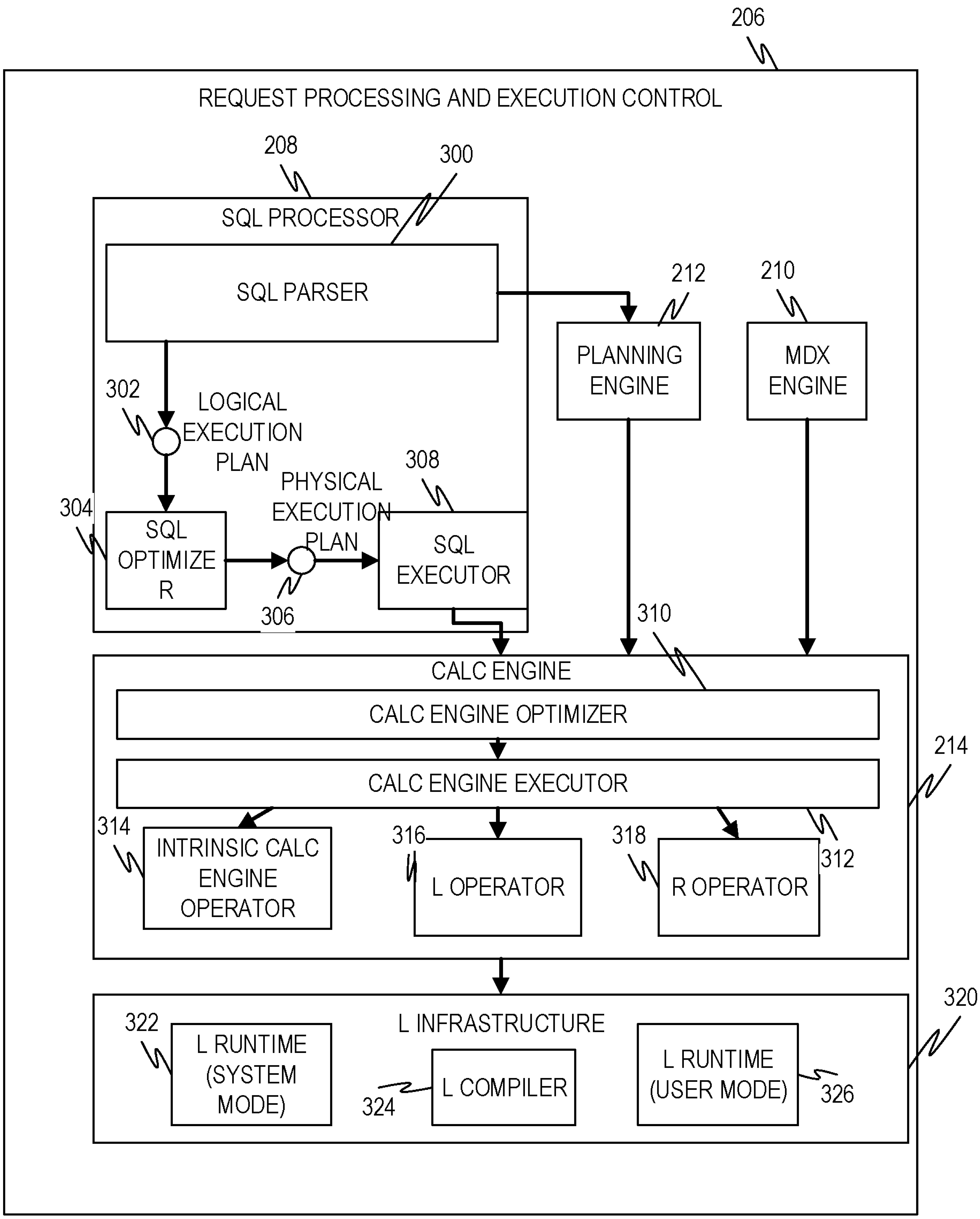

206
REQUEST PROCESSING AND EXECUTION CONTROL
208
300
SQL PROCESSOR
SQL PARSER
212
210
PLANNING ENGINE
MDX ENGINE
302
LOGICAL EXECUTION PLAN
308
PHYSICAL EXECUTION PLAN
304
SQL OPTIMIZE R
SQL EXECUTOR
306
310
CALC ENGINE
CALC ENGINE OPTIMIZER
214
CALC ENGINE EXECUTOR
314
INTRINSIC CALC ENGINE OPERATOR
316
L OPERATOR
318
R OPERATOR
312
320
322
L INFRASTRUCTURE
L RUNTIME (SYSTEM MODE)
L COMPILER
324
L RUNTIME (USER MODE)
326

*FIG. 3*

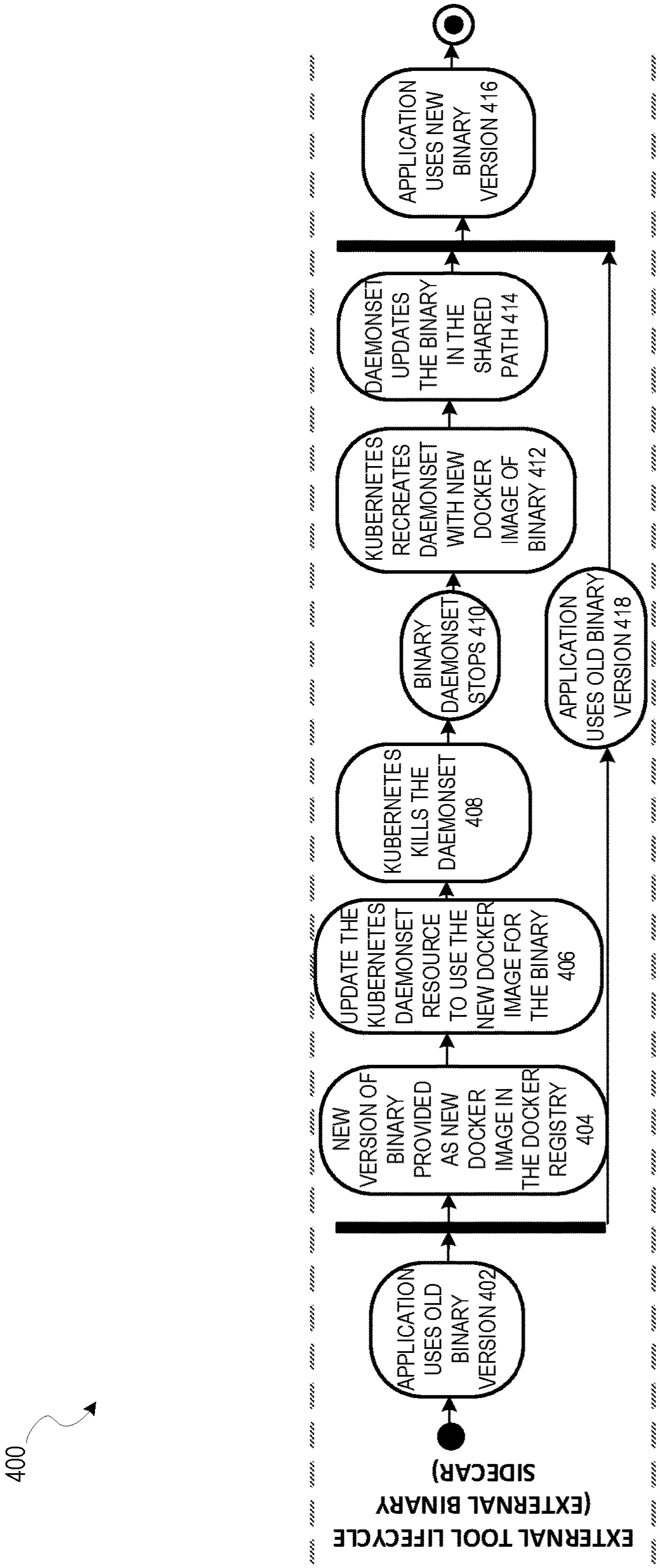

400
EXTERNAL TOOL LIFECYCLE (EXTERNAL BINARY SIDECAR)
APPLICATION USES OLD BINARY VERSION 402
NEW VERSION OF BINARY PROVIDED AS NEW DOCKER IMAGE IN THE DOCKER REGISTRY 404
UPDATE THE KUBERNETES DAEMONSET RESOURCE TO USE THE NEW DOCKER IMAGE FOR THE BINARY 406
KUBERNETES KILLS THE DAEMONSET 408
BINARY DAEMONSET STOPS 410
KUBERNETES RECREATES DAEMONSET WITH NEW DOCKER IMAGE OF BINARY 412
DAEMONSET UPDATES THE BINARY IN THE SHARED PATH 414
APPLICATION USES NEW BINARY VERSION 416
APPLICATION USES OLD BINARY VERSION 418

*FIG. 4*

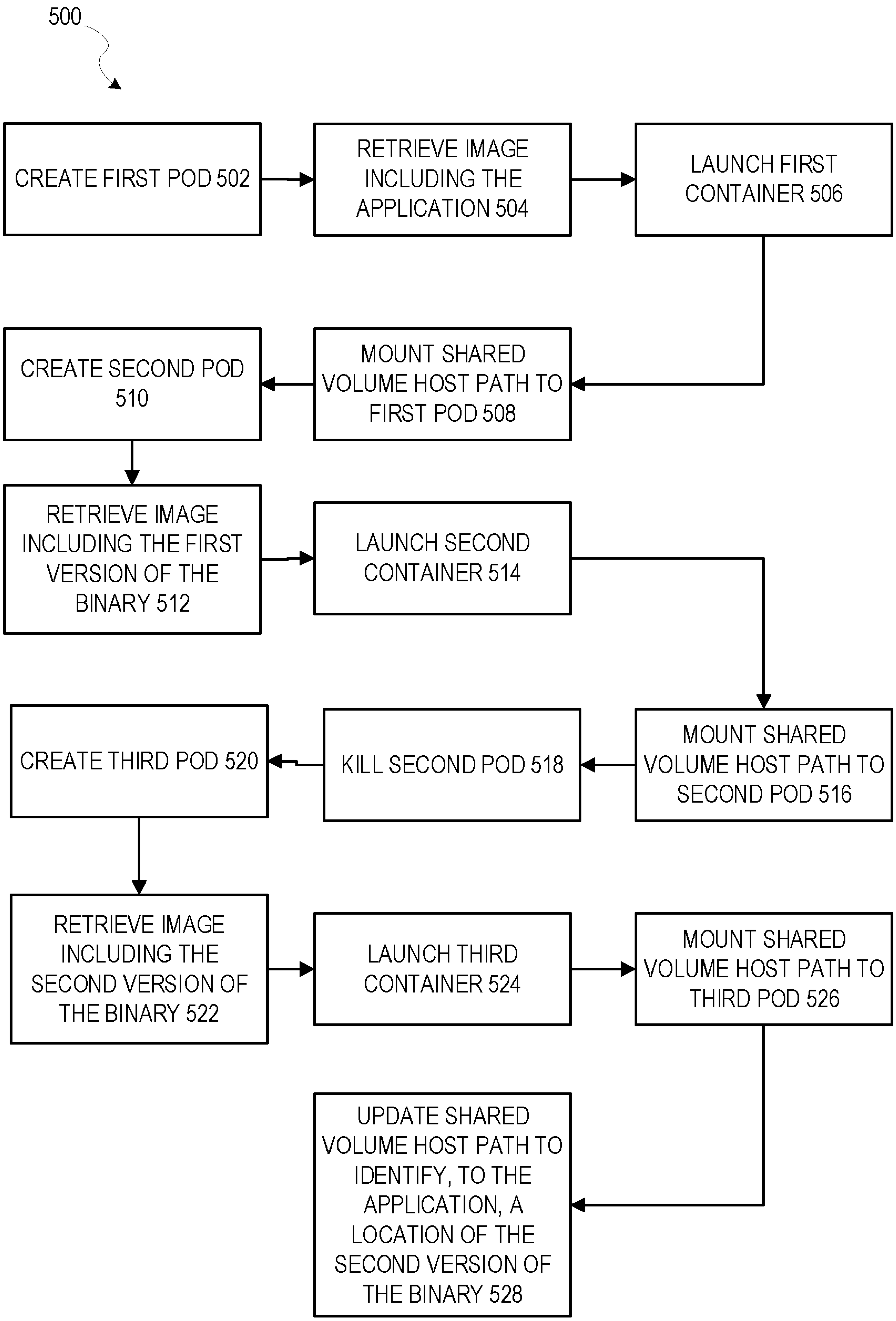

500
CREATE FIRST POD 502
RETRIEVE IMAGE INCLUDING THE APPLICATION 504
LAUNCH FIRST CONTAINER 506
MOUNT SHARED VOLUME HOST PATH TO FIRST POD 508
CREATE SECOND POD 510
RETRIEVE IMAGE INCLUDING THE FIRST VERSION OF THE BINARY 512
LAUNCH SECOND CONTAINER 514
MOUNT SHARED VOLUME HOST PATH TO SECOND POD 516
KILL SECOND POD 518
CREATE THIRD POD 520
RETRIEVE IMAGE INCLUDING THE SECOND VERSION OF THE BINARY 522
LAUNCH THIRD CONTAINER 524
MOUNT SHARED VOLUME HOST PATH TO THIRD POD 526
UPDATE SHARED VOLUME HOST PATH TO IDENTIFY, TO THE APPLICATION, A LOCATION OF THE SECOND VERSION OF THE BINARY 528

*FIG. 5*

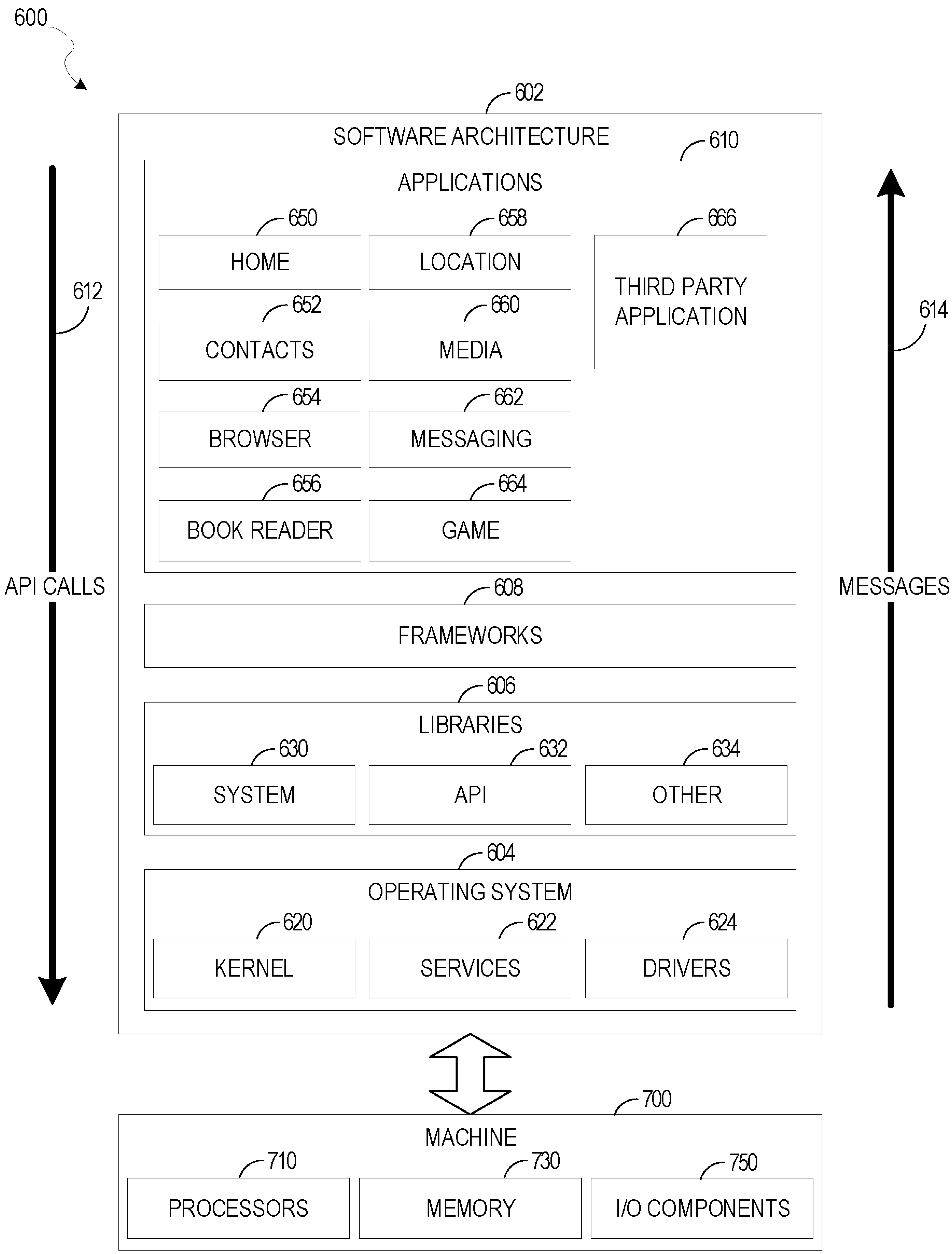

600
602
SOFTWARE ARCHITECTURE
610
APPLICATIONS
650
HOME
658
LOCATION
666
THIRD PARTY APPLICATION
652
CONTACTS
660
MEDIA
654
BROWSER
662
MESSAGING
656
BOOK READER
664
GAME
612
API CALLS
614
MESSAGES
608
FRAMEWORKS
606
LIBRARIES
630
SYSTEM
632
API
634
OTHER
604
OPERATING SYSTEM
620
KERNEL
622
SERVICES
624
DRIVERS
700
MACHINE
710
PROCESSORS
730
MEMORY
750
I/O COMPONENTS

*FIG. 6*

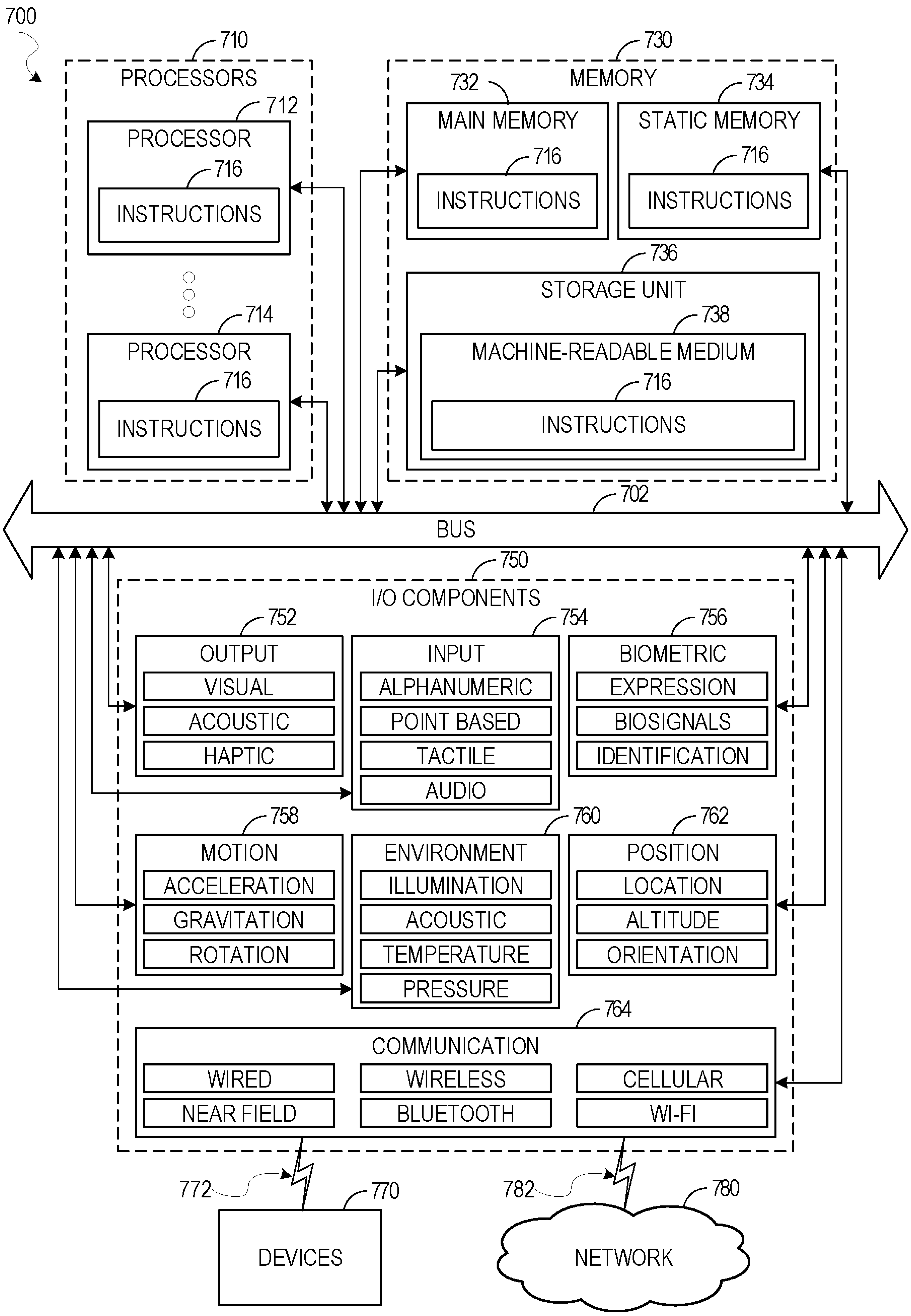

700
710
PROCESSORS
712
PROCESSOR
716
INSTRUCTIONS
714
PROCESSOR
716
INSTRUCTIONS
730
MEMORY
732
MAIN MEMORY
716
INSTRUCTIONS
734
STATIC MEMORY
716
INSTRUCTIONS
736
STORAGE UNIT
738
MACHINE-READABLE MEDIUM
716
INSTRUCTIONS
702
BUS
750
I/O COMPONENTS
752
OUTPUT
VISUAL
ACOUSTIC
HAPTIC
754
INPUT
ALPHANUMERIC
POINT BASED
TACTILE
AUDIO
756
BIOMETRIC
EXPRESSION
BIOSIGNALS
IDENTIFICATION
758
MOTION
ACCELERATION
GRAVITATION
ROTATION
760
ENVIRONMENT
ILLUMINATION
ACOUSTIC
TEMPERATURE
PRESSURE
762
POSITION
LOCATION
ALTITUDE
ORIENTATION
764
COMMUNICATION
WIRED
WIRELESS
CELLULAR
NEAR FIELD
BLUETOOTH
WI-FI
772
770
DEVICES
782
780
NETWORK

FIG. 7

# EXTERNAL BINARY SIDECAR FOR CLOUD CONTAINERS

## TECHNICAL FIELD

This document generally relates to in-memory database technology. More specifically, this document relates to an external binary sidecar for cloud containers for in-memory databases.

## BACKGROUND

An in-memory database (also known as an in-memory database management system) is a type of database management system that primarily relies on main memory for computer data storage. It is contrasted with database management systems that employ a disk storage mechanism. In-memory databases are traditionally faster than disk storage databases because disk access is slower than memory access. One example in-memory database is the HANA® database from SAP SE, of Walldorf, Germany.

## BRIEF DESCRIPTION OF DRAWINGS

The present disclosure is illustrated by way of example and not limitation in the figures of the accompanying drawings, in which like references indicate similar elements.

FIG. **1** is a diagram illustrating a cloud-based in-memory database management system, in accordance with an example embodiment.

FIG. **2** is a diagram illustrating an index server, in accordance with an example embodiment.

FIG. **3** is a diagram illustrating a request processing and execution control, in accordance with an example embodiment.

FIG. **4** is a sequence diagram illustrating a method of updating a binary without causing downtime in a corresponding application, in accordance with an example embodiment.

FIG. **5** is a flow diagram illustrating a method for creating a pod and updating a binary in accordance with an example embodiment.

FIG. **6** is a block diagram illustrating an architecture of software, which can be installed on any one or more of the devices described above.

FIG. **7** illustrates a diagrammatic representation of a machine in the form of a computer system within which a set of instructions may be executed for causing the machine to perform any one or more of the methodologies discussed herein, according to an example embodiment.

## DETAILED DESCRIPTION

The description that follows discusses illustrative systems, methods, techniques, instruction sequences, and computing machine program products. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide an understanding of various example embodiments of the present subject matter. It will be evident, however, to those skilled in the art, that various example embodiments of the present subject matter may be practiced without these specific details.

One implementation of in-memory databases is to place them in a cloud and allow access to the cloud database as a service. For the HANA® cloud, this is known as HANA®



as a service (HaaS). This allows customers to leverage the in-memory data processing and advanced analytic capabilities in the cloud.

Such services may allow applications to be developed using specialized in-memory database extended application services and deployed to a Cloud Foundry environment. Cloud Foundry is an open source, multi-cloud application platform as a service that allows for continuous delivery as it supports a full application development lifecycle, from initial deployment through testing stages to deployment. Cloud Foundry utilizes a container-based architecture that runs application in any programming language over a variety of cloud service providers.

It is common for an application running in an on-premises computer system to reference what is called a "binary". This binary is located in an accessible filesystem and is designed to perform a specific task. One example is that an in-memory database may need a binary to execute backups. Another example is that an in-memory database may need a special driver to connect to a data lake. These binaries are separate from the application and thus are often referred to as external binaries. Usually these external binaries are part of the delivery of the application and underlie the same lifecycle as the application. However, sometimes the external binary is updated without updating its corresponding application, and in an on-premises computer system it is possible to do so even when the application is running, without needing to restart the application

Such a process, however, is not currently possible inside a container-based architecture. Specifically, the external binary would need to be present in the container with the application, and updating the binary independently would mean needing to restart the container, which also restarts the running application.

What is needed is a solution that avoids these technical problems.

In an example embodiment, a solution is provided that causes a binary used by an in-memory database application to be deployed in a separate container. The container with the binary may be called a "sidecar" to the container with the in-memory database application. Both containers mount the same path in the hosts shared filesystem to expose the binary to the application. There is no permanent connection between the two containers, and thus the binary can be updated individually without inducing a downtime of the in-memory database application.

In an example embodiment, Kubernetes may be used as the container architecture. Kubernetes is a system for automating deployment, scaling, and management of containerized applications. Application containerization is a virtualization method used by operating systems to deploy and run distributed applications without launching an entire virtual machine for each application.

Containerized applications have advantages over standard applications. When a standard application is installed on a server, libraries required by the application are also installed. Thus, if multiple applications are installed, the libraries on the server are an amalgamation of the libraries required by each of the multiple applications. If one application installs a different version of a library used by another application, the first installed version is overwritten. As a result, an application may use a version of a library that was not tested with the application, which may further result in unexpected behavior.

Kubernetes containers, by virtue of being so modular, are quite conducive to scaling of in-memory database instances. Kubernetes containers are called pods. Each pod is scheduled on a specific host. If the host becomes unavailable, Kubernetes automatically instantiates the instance on a different host, greatly easing maintenance.

FIG. **1** is a diagram illustrating a cloud-based in-memory database management system **100**, including its client/external connection points, which can be kept stable in the case of disaster recovery to ensure stable service operations, in accordance with an example embodiment. Here, the in-memory database management system **100** may be coupled to one or more client applications **102**A, **102**B. The client applications **102**A, **102**B may communicate with the in-memory database management system **100** through a number of different protocols, including Structured Query Language (SQL), Multidimensional Expressions (MDX), Hypertext Transfer Protocol (HTTP), REST, and Hypertext Markup Language (HTML).

The in-memory database management system **100** may be stored in cloud cluster **104**. Also depicted is a cloud cockpit **106**, used to perform lifecycle operations on the in-memory database management system **100**. The one or more client applications **102**A, **102**B may access the in-memory database system via an edge router **108**. On the back end, however, Kubernetes **110** is used to manage the specific instance of the in-memory database, which may be stored in a first Kubernetes pod **112** or other container running on a host **113**.

An in-memory database operator **114** may receive a request from the cloud cockpit **106** to create an instance of the in-memory database management system **100**. This request may include, or at least reference, an in-memory database resource **114**, which specifies one or more requirements of the in-memory database management system **100**. The in-memory database operator **114** may then interface with Kubernetes **110** to create the in-memory database management system **100** in the first pod **112**. More particularly, the in-memory database resource may be established by a service broker application program interface (API).

The in-memory database management system **100** may comprise a number of different components, including an index server **116**, an XS engine **118**, a statistics server **120**, a preprocessor server **122**, and a name server **124**. These components may operate on a single computing device.

The index server **116** contains the actual data and the engines for processing the data. It also coordinates and uses all the other servers.

The XS engine **118** allows clients to connect to the in-memory database management system **100** using web protocols, such as HTTP.

The statistics server **120** collects information about status, performance, and resource consumption from all the other server components. The statistics server **120** can be accessed from the cloud cockpit **106** to obtain the status of various alert monitors.

The preprocessor server **122** is used for analyzing text data and extracting the information on which text search capabilities are based.

The name server **124** holds information about the database topology. This is used in a distributed system with instances of the database on different hosts. The name server **124** knows where the components are running and which data is located on which server.

In an example embodiment, another Kubernetes operator, which may be called a binary operator **125** creates a second Kubernetes pod **126** using a binary resource **127**. The second Kubernetes pod **126** contains a binary **128**. This binary **128** may be, for example, a third-party tool that may be utilized by the instance of the in-memory database management system **100** when needed and may perform a task that the in-memory database management system **100** could not itself perform, such as backing the in-memory database management system **100** up. As will be described later, the binary operator **125** may be designed such that it uses a specialized procedure that allows the binary **128** to be updated during the running of the in-memory database management system **100**, without requiring any downtime or restart of the in-memory database management system **100**. A shared volume host path **130** may be used to allow the in-memory database management system **100** to access the binary **128**. It should be noted that, while not pictured, more than one instance of the in-memory database **100** may be running.

FIG. **2** is a diagram illustrating an index server **116**, in accordance with an example embodiment. Specifically, the index server **116** of FIG. **1** is depicted in more detail. The index server **116** includes a connection and session management component **200**, which is responsible for creating and managing sessions and connections for the database clients. Once a session is established, clients can communicate with the database system using SQL statements. For each session, a set of session parameters **202** may be maintained, such as auto-commit, current transaction isolation level, etc. Users (e.g., system administrators, developers) may be authenticated by the database system itself (e.g., by logging in with user name and password, using an authentication component **204**), or authentication can be delegated to an external authentication provider such as a Lightweight Directory Access Protocol (LDAP) directory.

The client requests can be analyzed and executed by a set of components summarized as request processing and execution control **206**. An SQL processor **208** checks the syntax and semantics of the client SQL statements and generates a logical execution plan. Multidimensional expressions (MDX) are a language for querying and manipulating multidimensional data stored in online analytical processing (OLAP) cubes. As such, an MDX engine **210** is provided to allow for the parsing and executing of MDX commands. A planning engine **212** allows applications (e.g., financial planning applications) to execute basic planning operations in the database layer. One such operation is to create a new version of a dataset as a copy of an existing dataset, while applying filters and transformations.

A calc engine **214** implements the various SQL script and planning operations. The calc engine **214** creates a logical execution plan for calculation models derived from SQL scripts, MDX, planning, and domain-specific models. This logical execution plan may include, for example, breaking up a model into operations that can be processed in parallel.

The data is stored in relational stores **216**, which implement a relational database in main memory.

Each SQL statement may be processed in the context of a transaction. New sessions are implicitly assigned to a new transaction. A transaction manager **218** coordinates database transactions, controls transactional isolation, and keeps track of running and closed transactions. When a transaction is committed or rolled back, the transaction manager **218** informs the involved engines about this event so they can execute needed actions. The transaction manager **218** also cooperates with a persistence layer **220** to achieve atomic and durable transactions.

An authorization manager **222** is invoked by other database system components to check whether the user has the specified privileges to execute the requested operations. The database system allows for the granting of privileges to users or roles. A privilege grants the right to perform a specified operation on a specified object.

The persistence layer **220** ensures that the database is restored to the most recent committed state after a restart and that transactions are either completely executed or completely undone. To achieve this goal in an efficient way, the persistence layer **220** uses a combination of write-ahead logs, shadow paging, and save points. The persistence layer **220** also offers a page management interface **224** for writing and reading data to and from a separate disk storage **226**, and also contains a logger **228** that manages the transaction log. Log entries can be written implicitly by the persistence layer **220** when data is written via the persistence interface or explicitly by using a log interface.

FIG. **3** is a diagram illustrating a request processing and execution control **206**, in accordance with an example embodiment. This diagram depicts the request processing and execution control **206** of FIG. **2** in more detail. The SQL processor **208** contains an SQL parser **300**, which parses an SQL statement and generates a logical execution plan **302**, which it passes to an SQL optimizer **304**. The SQL optimizer **304** then optimizes the logical execution plan **302** and converts it to a physical execution plan **306**, which it then passes to an SQL executor **308**. The intrinsic calc engine operator **314** implements the various SQL script and planning operations, and includes a calc engine optimizer **310**, which optimizes the operations, and a calc engine executor **312**, which executes the operations, as well as an intrinsic calc engine operator **314**, an L operator **316**, and an R operator **318**.

An L infrastructure **320** includes a number of components to aid in the running of L procedures, including an L-runtime (system mode) **322**, an L compiler **324**, and an L-runtime (user mode) **326**.

As described earlier, an operator acts to create the instance of the in-memory database in a first Kubernetes pod **112** and a second operator creates a binary **128** in a second Kubernetes pod **126**.

In one embodiment, the container framework is a Docker framework. Other types of container frameworks may also be useful. For example, container frameworks such as LXC or Rocket container frameworks may also be useful. Docker, for example, may be the format of the image to be run in a container runtime.

FIG. **4** is a sequence diagram illustrating a method **400** of updating a binary without causing downtime in a corresponding application, in accordance with an example embodiment. At operation **402**, the application is operating using an old version of the binary. At operation **404**, the new version of the binary is provided. This may be provided, for example, as a new Docker or other container-based service image in a Docker registry. At operation **406**, a DaemonSet Kubernetes resource is updated to use the new Docker image for the binary. A DaemonSet resource is a resource that ensures that multiple nodes are able to run a copy of a pod. As nodes are added to a cluster, pods are added to them. As nodes are removed from a cluster, garbage collection is performed on the pods. A DaemonSet may be used, for example, for running a cluster storage daemon, running a log collection daemon on each node, and running a node monitoring daemon ion each node. At operation **408**, Kubernetes kills the DaemonSet. At operation **410**, the binary DaemonSet stops. At operation **412**, Kubernetes recreates the DaemonSet with the new Docker image of the binary. At operation **414**, DaemonSet updates the binary in the shared volume (host path). At this point, at operation **416**, the application uses the new binary version. It should be noted that operation **418**, where the application uses the old binary version, actually continues throughout the performance of operations **404**-**416**, allowing the old version of the binary to be used up until the point that the new binary version is ready to be used, without causing any downtime in the application.

It should be noted that the application and the binary are deployed in separate containers in separate Kubernetes resources. This is in contrast to a solution where the application and binary are deployed in separate containers on the same Kubernetes pod with a shared file system. Such a solution would still make use of application downtime when using a new binary version, because the Kubernetes pod with the new Docker image of the binary would need to be recreated and during this recreation time the pod would not be operating.

FIG. **5** is a flow diagram illustrating a method **500** for creating a pod and updating a binary in accordance with an example embodiment. At operation **502**, a first pod in a container-orchestration system is created. The first pod contains a reference to an image including an application. The first pod may then be run, which may cause operations **504**-**510** to be performed. At operation **504**, the image including the application is retrieved. At operation **506**, a first container in a container-based virtualization system is launched. The application gets started during this launch. At operation **508**, a shared volume host path is mounted to the first pod.

At operation **510**, a second pod in a container-orchestration system is created. The second pod contains a reference to an image including a first version of the binary associated with the application, wherein the application contains one or more calls that, when executed, cause the binary to perform one or more operations. The second pod may then be run, which may cause operations **512**-**518** to be performed. At operation **512**, the image including the first version of the binary is retrieved. At operation **514**, a second container in the container-based virtualization system is launched. At operation **516**, the shared volume host path is mounted to the second pod. At this point the shared volume host path identifies, to the application, a location of the first version of the binary.

The application is then able to operate using the first version of the binary. Operations **518**-**528** may then be performed to allow the application to utilize a second version of the binary. While operations **518**-**528** are being performed, however, the application may continue to utilize the first version of the binary.

At operation **518**, the second pod is killed. Killing a pod involves stopping its execution as well as optionally deleting the pod.

At operation **520**, a third pod in a container-orchestration system is created. The third pod contains a reference to an image including a second version of the binary associated with the application. The third pod may then be run, which may cause operations **522**-**528** to be performed. At operation **522**, the image including the second version of the binary is retrieved. At operation **524**, a third container in the container-based virtualization system is launched. At operation **526**, the shared volume host path is mounted to the third pod.

At operation **528**, the shared volume host path is updated to identify, to the application, a location of the second version of the binary.

Once this is performed, the application is then able to use the second version of the binary, without any downtime in the application having been needed due to the update of the first version of the binary to the second version of the binary.

EXAMPLES

Example 1

A system comprising:
at least one hardware processor; and
a computer-readable medium storing instructions that, when executed by the at least one hardware processor, cause the at least one hardware processor to perform operations comprising:
creating a first pod in a container-orchestration system, the first pod containing a reference to an image including an application;
running the first pod on a first host, the running including:
retrieving the image including the application;
launching a first container in a container-based virtualization system;
starting the application; and
mounting a shared volume host path to the first pod.
creating a second pod in a container-orchestration system, the second pod containing a reference to an image including a first version of the binary associated with the application, wherein the application contains one or more calls that, when executed, cause the binary to perform one or more operations;
running the second pod on the first host, the running including:
retrieving the image including the first version of the binary;
launching a second container in the container-based virtualization system; and
mounting the shared volume host path to the second pod, the shared volume host path identifying, to the application, a location of the first version of the binary.

Example 2

The system of Example 1, wherein the operations further comprise:
creating a third pod in a container-orchestration system, the third pod containing a reference to an image including a second version of the binary associated with the application;
killing the second pod;
running the third pod on the first host, the running including:
retrieving the image including the second version of the binary;
launching a third container in the container-based virtualization system;
mounting the shared volume host path to the third pod; and
updating the shared volume host path to identify, to the application, a location of the second version of the binary.

Example 3

The system of Examples 1 or 2, wherein the application is an in-memory database.

Example 4

The system of Example 3, wherein the binary is an executable to perform backups of the in-memory database.

Example 5

The system of Example 3, wherein the binary is a driver to connect the in-memory database to a data lake.

Example 6

The system of any of Examples 1-5, wherein the container-based virtualization system is Docker.

Example 7

The system of any of Examples 1-6, wherein the container-orchestration system is Kubernetes.

Example 8

A method comprising:
creating a first pod in a container-orchestration system, the first pod containing a reference to an image including an application;
running the first pod on a first host, the running including:
retrieving the image including the application;
launching a first container in a container-based virtualization system;
starting the application; and
mounting a shared volume host path to the first pod.
creating a second pod in a container-orchestration system, the second pod containing a reference to an image including a first version of the binary associated with the application, wherein the application contains one or more calls that, when executed, cause the binary to perform one or more operations;
running the second pod on the first host, the running including:
retrieving the image including the first version of the binary;
launching a second container in the container-based virtualization system; and
mounting the shared volume host path to the second pod, the shared volume host path identifying, to the application, a location of the first version of the binary.

Example 9

The method of Example 8, wherein the operations further comprise:
creating a third pod in a container-orchestration system, the third pod containing a reference to an image including a second version of the binary associated with the application;
killing the second pod;
running the third pod on the first host, the running including:
retrieving the image including the second version of the binary;
launching a third container in the container-based virtualization system;
mounting the shared volume host path to the third pod; and

updating the shared volume host path to identify, to the application, a location of the second version of the binary.

Example 10

The method of Examples 8 or 9, wherein the application is an in-memory database.

Example 11

The method of Example 10, wherein the binary is an executable to perform backups of the in-memory database.

Example 12

The method of Example 10, wherein the binary is a driver to connect the in-memory database to a data lake.

Example 13

The method of any of Examples 8-12, wherein the container-based virtualization system is Docker.

Example 14

The method of any of Examples 8-13, wherein the container-orchestration system is Kubernetes.

Example 15

A non-transitory machine-readable medium storing instructions which, when executed by one or more processors, cause the one or more processors to perform operations comprising:

creating a first pod in a container-orchestration system, the first pod containing a reference to an image including an application;

running the first pod on a first host, the running including:

retrieving the image including the application;

launching a first container in a container-based virtualization system;

starting the application; and

mounting a shared volume host path to the first pod.

creating a second pod in a container-orchestration system, the second pod containing a reference to an image including a first version of the binary associated with the application, wherein the application contains one or more calls that, when executed, cause the binary to perform one or more operations;

running the second pod on the first host, the running including:

retrieving the image including the first version of the binary;

launching a second container in the container-based virtualization system; and

mounting the shared volume host path to the second pod, the shared volume host path identifying, to the application, a location of the first version of the binary.

Example 16

The non-transitory machine-readable medium of Example 15, wherein the operations further comprise:

creating a third pod in a container-orchestration system, the third pod containing a reference to an image including a second version of the binary associated with the application;

killing the second pod;

running the third pod on the first host, the running including:

retrieving the image including the second version of the binary;

launching a third container in the container-based virtualization system;

mounting the shared volume host path to the third pod; and

updating the shared volume host path to identify, to the application, a location of the second version of the binary.

Example 17

The non-transitory machine-readable medium of Examples 15 or 16, wherein the application is an in-memory database.

Example 18

The non-transitory machine-readable medium of Example 17, wherein the binary is an executable to perform backups of the in-memory database.

Example 19

The non-transitory machine-readable medium of Example 17, wherein the binary is a driver to connect the in-memory database to a data lake.

Example 20

The non-transitory machine-readable medium of any of Examples 15-19, wherein the container-based virtualization system is Docker.

FIG. **6** is a block diagram **600** illustrating a software architecture **602**, which can be installed on any one or more of the devices described above. FIG. **6** is merely a non-limiting example of a software architecture, and it will be appreciated that many other architectures can be implemented to facilitate the functionality described herein. In various embodiments, the software architecture **602** is implemented by hardware such as a machine **700** of FIG. **7** that includes processors **710**, memory **730**, and input/output (I/O) components **750**. In this example architecture, the software architecture **602** can be conceptualized as a stack of layers where each layer may provide a particular functionality. For example, the software architecture **602** includes layers such as an operating system **604**, libraries **606**, frameworks **608**, and applications **610**. Operationally, the applications **610** invoke API calls **612** through the software stack and receive messages **614** in response to the API calls **612**, consistent with some embodiments.

In various implementations, the operating system **604** manages hardware resources and provides common services. The operating system **604** includes, for example, a kernel **620**, services **622**, and drivers **624**. The kernel **620** acts as an abstraction layer between the hardware and the other software layers, consistent with some embodiments. For example, the kernel **620** provides memory management, processor management (e.g., scheduling), component management, networking, and security settings, among other functionality. The services **622** can provide other common services for the other software layers. The drivers **624** are responsible for controlling or interfacing with the underlying hardware, according to some embodiments. For instance, the drivers **624** can include display drivers, camera drivers, BLUETOOTH® or BLUETOOTH® Low-Energy drivers, flash memory drivers, serial communication drivers (e.g., Universal Serial Bus (USB) drivers), Wi-Fi® drivers, audio drivers, power management drivers, and so forth.

In some embodiments, the libraries **606** provide a low-level common infrastructure utilized by the applications **610**. The libraries **606** can include system libraries **630** (e.g., C standard library) that can provide functions such as memory allocation functions, string manipulation functions, mathematic functions, and the like. In addition, the libraries **606** can include API libraries **632** such as media libraries (e.g., libraries to support presentation and manipulation of various media formats such as Moving Picture Experts Group-4 (MPEG4), Advanced Video Coding (H.264 or AVC), Moving Picture Experts Group Layer-3 (MP3), Advanced Audio Coding (AAC), Adaptive Multi-Rate (AMR) audio codec, Joint Photographic Experts Group (JPEG or JPG), or Portable Network Graphics (PNG)), graphics libraries (e.g., an OpenGL framework used to render in 2D and 3D in a graphic context on a display), database libraries (e.g., SQLite to provide various relational database functions), web libraries (e.g., WebKit to provide web browsing functionality), and the like. The libraries **606** can also include a wide variety of other libraries **634** to provide many other APIs to the applications **610**.

The frameworks **608** provide a high-level common infrastructure that can be utilized by the applications **610**, according to some embodiments. For example, the frameworks **608** provide various graphical user interface (GUI) functions, high-level resource management, high-level location services, and so forth. The frameworks **608** can provide a broad spectrum of other APIs that can be utilized by the applications **610**, some of which may be specific to a particular operating system **604** or platform.

In an example embodiment, the applications **610** include a home application **650**, a contacts application **652**, a browser application **654**, a book reader application **656**, a location application **658**, a media application **660**, a messaging application **662**, a game application **664**, and a broad assortment of other applications, such as a third-party application **666**. According to some embodiments, the applications **610** are programs that execute functions defined in the programs. Various programming languages can be employed to create one or more of the applications **610**, structured in a variety of manners, such as object-oriented programming languages (e.g., Objective-C, Java, or C++) or procedural programming languages (e.g., C or assembly language). In a specific example, the third-party application **666** (e.g., an application developed using the ANDROID™ or IOS™ software development kit (SDK) by an entity other than the vendor of the particular platform) may be mobile software running on a mobile operating system such as IOS™, ANDROID™, WINDOWS® Phone, or another mobile operating system. In this example, the third-party application **666** can invoke the API calls **612** provided by the operating system **604** to facilitate functionality described herein.

FIG. **7** illustrates a diagrammatic representation of a machine **700** in the form of a computer system within which a set of instructions may be executed for causing the machine **700** to perform any one or more of the methodologies discussed herein, according to an example embodiment. Specifically, FIG. **7** shows a diagrammatic representation of the machine **700** in the example form of a computer system, within which instructions **716** (e.g., software, a program, an application, an applet, an app, or other executable code) for causing the machine **700** to perform any one or more of the methodologies discussed herein may be executed. For example, the instructions **716** may cause the machine **700** to execute the methods of FIG. **5**. Additionally, or alternatively, the instructions **716** may implement FIGS. **1-5** and so forth. The instructions **716** transform the general, non-programmed machine **700** into a particular machine **700** programmed to carry out the described and illustrated functions in the manner described. In alternative embodiments, the machine **700** operates as a standalone device or may be coupled (e.g., networked) to other machines. In a networked deployment, the machine **700** may operate in the capacity of a server machine or a client machine in a server-client network environment, or as a peer machine in a peer-to-peer (or distributed) network environment. The machine **700** may comprise, but not be limited to, a server computer, a client computer, a personal computer (PC), a tablet computer, a laptop computer, a netbook, a set-top box (STB), a personal digital assistant (PDA), an entertainment media system, a cellular telephone, a smart phone, a mobile device, a wearable device (e.g., a smart watch), a smart home device (e.g., a smart appliance), other smart devices, a web appliance, a network router, a network switch, a network bridge, or any machine capable of executing the instructions **716**, sequentially or otherwise, that specify actions to be taken by the machine **700**. Further, while only a single machine **700** is illustrated, the term "machine" shall also be taken to include a collection of machines **700** that individually or jointly execute the instructions **716** to perform any one or more of the methodologies discussed herein.

The machine **700** may include processors **710**, memory **730**, and I/O components **750**, which may be configured to communicate with each other such as via a bus **702**. In an example embodiment, the processors **710** (e.g., a central processing unit (CPU), a reduced instruction set computing (RISC) processor, a complex instruction set computing (CISC) processor, a graphics processing unit (GPU), a digital signal processor (DSP), an application-specific integrated circuit (ASIC), a radio-frequency integrated circuit (RFIC), another processor, or any suitable combination thereof) may include, for example, a processor **712** and a processor **714** that may execute the instructions **716**. The term "processor" is intended to include multi-core processors that may comprise two or more independent processors (sometimes referred to as "cores") that may execute instructions **716** contemporaneously. Although FIG. **7** shows multiple processors **710**, the machine **700** may include a single processor **712** with a single core, a single processor **712** with multiple cores (e.g., a multi-core processor **712**), multiple processors **712**, **714** with a single core, multiple processors **712**, **714** with multiple cores, or any combination thereof.

The memory **730** may include a main memory **732**, a static memory **734**, and a storage unit **736**, each accessible to the processors **710** such as via the bus **702**. The main memory **732**, the static memory **734**, and the storage unit **736** store the instructions **716** embodying any one or more of the methodologies or functions described herein. The instructions **716** may also reside, completely or partially, within the main memory **732**, within the static memory **734**, within the storage unit **736**, within at least one of the processors **710** (e.g., within the processor's cache memory), or any suitable combination thereof, during execution thereof by the machine **700**.

The I/O components **750** may include a wide variety of components to receive input, provide output, produce output, transmit information, exchange information, capture measurements, and so on. The specific I/O components **750** that are included in a particular machine will depend on the type of machine. For example, portable machines such as mobile phones will likely include a touch input device or other such input mechanisms, while a headless server machine will likely not include such a touch input device. It will be appreciated that the I/O components **750** may include many other components that are not shown in FIG. **7**. The I/O components **750** are grouped according to functionality merely for simplifying the following discussion, and the grouping is in no way limiting. In various example embodiments, the I/O components **750** may include output components **752** and input components **754**. The output components **752** may include visual components (e.g., a display such as a plasma display panel (PDP), a light-emitting diode (LED) display, a liquid crystal display (LCD), a projector, or a cathode ray tube (CRT)), acoustic components (e.g., speakers), haptic components (e.g., a vibratory motor, resistance mechanisms), other signal generators, and so forth. The input components **754** may include alphanumeric input components (e.g., a keyboard, a touch screen configured to receive alphanumeric input, a photo-optical keyboard, or other alphanumeric input components), point-based input components (e.g., a mouse, a touchpad, a trackball, a joystick, a motion sensor, or another pointing instrument), tactile input components (e.g., a physical button, a touch screen that provides location and/or force of touches or touch gestures, or other tactile input components), audio input components (e.g., a microphone), and the like.

In further example embodiments, the I/O components **750** may include biometric components **756**, motion components **758**, environmental components **760**, or position components **762**, among a wide array of other components. For example, the biometric components **756** may include components to detect expressions (e.g., hand expressions, facial expressions, vocal expressions, body gestures, or eye tracking), measure biosignals (e.g., blood pressure, heart rate, body temperature, perspiration, or brain waves), identify a person (e.g., voice identification, retinal identification, facial identification, fingerprint identification, or electroencephalogram-based identification), and the like. The motion components **758** may include acceleration sensor components (e.g., accelerometer), gravitation sensor components, rotation sensor components (e.g., gyroscope), and so forth. The environmental components **760** may include, for example, illumination sensor components (e.g., photometer), temperature sensor components (e.g., one or more thermometers that detect ambient temperature), humidity sensor components, pressure sensor components (e.g., barometer), acoustic sensor components (e.g., one or more microphones that detect background noise), proximity sensor components (e.g., infrared sensors that detect nearby objects), gas sensors (e.g., gas detection sensors to detect concentrations of hazardous gases for safety or to measure pollutants in the atmosphere), or other components that may provide indications, measurements, or signals corresponding to a surrounding physical environment. The position components **762** may include location sensor components (e.g., a Global Positioning System (GPS) receiver component), altitude sensor components (e.g., altimeters or barometers that detect air pressure from which altitude may be derived), orientation sensor components (e.g., magnetometers), and the like.

Communication may be implemented using a wide variety of technologies. The I/O components **750** may include communication components **764** operable to couple the machine **700** to a network **780** or devices **770** via a coupling **782** and a coupling **772**, respectively. For example, the communication components **764** may include a network interface component or another suitable device to interface with the network **780**. In further examples, the communication components **764** may include wired communication components, wireless communication components, cellular communication components, near field communication (NFC) components, Bluetooth® components (e.g., Bluetooth® Low Energy), Wi-Fi® components, and other communication components to provide communication via other modalities. The devices **770** may be another machine or any of a wide variety of peripheral devices (e.g., coupled via a USB).

Moreover, the communication components **764** may detect identifiers or include components operable to detect identifiers. For example, the communication components **764** may include radio-frequency identification (RFID) tag reader components, NFC smart tag detection components, optical reader components (e.g., an optical sensor to detect one-dimensional bar codes such as Universal Product Code (UPC) bar code, multi-dimensional bar codes such as QR code, Aztec code, Data Matrix, Dataglyph, MaxiCode, PDF417, Ultra Code, UCC RSS-2D bar code, and other optical codes), or acoustic detection components (e.g., microphones to identify tagged audio signals). In addition, a variety of information may be derived via the communication components **764**, such as location via Internet Protocol (IP) geolocation, location via Wi-Fi® signal triangulation, location via detecting an NFC beacon signal that may indicate a particular location, and so forth.

The various memories (i.e., **730**, **732**, **734**, and/or memory of the processor(s) **710**) and/or the storage unit **736** may store one or more sets of instructions **716** and data structures (e.g., software) embodying or utilized by any one or more of the methodologies or functions described herein. These instructions (e.g., the instructions **716**), when executed by the processor(s) **710**, cause various operations to implement the disclosed embodiments.

As used herein, the terms "machine-storage medium," "device-storage medium," and "computer-storage medium" mean the same thing and may be used interchangeably. The terms refer to a single or multiple storage devices and/or media (e.g., a centralized or distributed database, and/or associated caches and servers) that store executable instructions and/or data. The terms shall accordingly be taken to include, but not be limited to, solid-state memories, and optical and magnetic media, including memory internal or external to processors. Specific examples of machine-storage media, computer-storage media, and/or device-storage media include non-volatile memory, including by way of example semiconductor memory devices, e.g., erasable programmable read-only memory (EPROM), electrically erasable programmable read-only memory (EEPROM), field-programmable gate array (FPGA), and flash memory devices; magnetic disks such as internal hard disks and removable disks; magneto-optical disks; and CD-ROM and DVD-ROM disks. The terms "machine-storage media," "computer-storage media," and "device-storage media" specifically exclude carrier waves, modulated data signals, and other such media, at least some of which are covered under the term "signal medium" discussed below.

In various example embodiments, one or more portions of the network **780** may be an ad hoc network, an intranet, an extranet, a virtual private network (VPN), a local-area network (LAN), a wireless LAN (WLAN), a wide-area network (WAN), a wireless WAN (WWAN), a metropolitan-area network (MAN), the Internet, a portion of the Internet, a portion of the public switched telephone network (PSTN), a plain old telephone service (POTS) network, a cellular telephone network, a wireless network, a Wi-Fi® network, another type of network, or a combination of two or more such networks. For example, the network **780** or a portion of the network **780** may include a wireless or cellular network, and the coupling **782** may be a Code Division Multiple Access (CDMA) connection, a Global System for Mobile communications (GSM) connection, or another type of cellular or wireless coupling. In this example, the coupling **782** may implement any of a variety of types of data transfer technology, such as Single Carrier Radio Transmission Technology (1×RTT), Evolution-Data Optimized (EVDO) technology, General Packet Radio Service (GPRS) technology, Enhanced Data rates for GSM Evolution (EDGE) technology, third Generation Partnership Project (3GPP) including 3G, fourth generation wireless (4G) networks, Universal Mobile Telecommunications System (UMTS), High-Speed Packet Access (HSPA), Worldwide Interoperability for Microwave Access (WiMAX), Long-Term Evolution (LTE) standard, others defined by various standard-setting organizations, other long-range protocols, or other data transfer technology.

The instructions **716** may be transmitted or received over the network **780** using a transmission medium via a network interface device (e.g., a network interface component included in the communication components **764**) and utilizing any one of a number of well-known transfer protocols (e.g., Hypertext Transfer Protocol (HTTP)). Similarly, the instructions **716** may be transmitted or received using a transmission medium via the coupling **772** (e.g., a peer-to-peer coupling) to the devices **770**. The terms "transmission medium" and "signal medium" mean the same thing and may be used interchangeably in this disclosure. The terms "transmission medium" and "signal medium" shall be taken to include any intangible medium that is capable of storing, encoding, or carrying the instructions **716** for execution by the machine **700**, and include digital or analog communications signals or other intangible media to facilitate communication of such software. Hence, the terms "transmission medium" and "signal medium" shall be taken to include any form of modulated data signal, carrier wave, and so forth. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal.

The terms "machine-readable medium," "computer-readable medium," and "device-readable medium" mean the same thing and may be used interchangeably in this disclosure. The terms are defined to include both machine-storage media and transmission media. Thus, the terms include both storage devices/media and carrier waves/modulated data signals.

What is claimed is:

**1**. A system comprising:
at least one hardware processor; and
a computer-readable medium storing instructions that, when executed by the at least one hardware processor, cause the at least one hardware processor to perform operations comprising:
creating a first pod in a container-orchestration system, the first pod containing a reference to an image including an application;
running the first pod on a first host, the running including:
retrieving the image including the application;
launching a first container in a container-based virtualization system;
starting the application; and
mounting a shared volume host path to the first pod;
creating a second pod in a container-orchestration system, the second pod containing a reference to an image including a first version of the binary associated with the application, wherein the application contains one or more calls that, when executed, cause the binary to perform one or more operations;
running the second pod on the first host, the running including:
retrieving the image including the first version of the binary;
launching a second container in the container-based virtualization system; and
mounting the shared volume host path to the second pod, the shared volume host path identifying, to the application, a location of the first version of the binary.

**2**. The system of claim **1**, wherein the operations further comprise:
creating a third pod in a container-orchestration system, the third pod containing a reference to an image including a second version of the binary associated with the application;
killing the second pod;
running the third pod on the first host, the running including:
retrieving the image including the second version of the binary;
launching a third container in the container-based virtualization system;
mounting the shared volume host path to the third pod; and
updating the shared volume host path to identify, to the application, a location of the second version of the binary.

**3**. The system of claim **1**, wherein the application is an in-memory database.

**4**. The system of claim **3**, wherein the binary is an executable to perform backups of the in-memory database.

**5**. The system of claim **3**, wherein the binary is a driver to connect the in-memory database to a data lake.

**6**. The system of claim **1**, wherein the container-based virtualization system is Docker.

**7**. The system of claim **1**, wherein the container-orchestration system is Kubernetes.

**8**. A method comprising:
creating a first pod in a container-orchestration system, the first pod containing a reference to an image including an application;
running the first pod on a first host, the running including:
retrieving the image including the application;
launching a first container in a container-based virtualization system;
starting the application; and
mounting a shared volume host path to the first pod;
creating a second pod in a container-orchestration system, the second pod containing a reference to an image including a first version of the binary associated with the application, wherein the application contains one or more calls that, when executed, cause the binary to perform one or more operations;
running the second pod on the first host, the running including:

retrieving the image including the first version of the binary;

launching a second container in the container-based virtualization system; and

mounting the shared volume host path to the second pod, the shared volume host path identifying, to the application, a location of the first version of the binary.

9. The method of claim **8**, wherein the operations further comprise:

creating a third pod in a container-orchestration system, the third pod containing a reference to an image including a second version of the binary associated with the application;

killing the second pod;

running the third pod on the first host, the running including:

retrieving the image including the second version of the binary;

launching a third container in the container-based virtualization system;

mounting the shared volume host path to the third pod; and

updating the shared volume host path to identify, to the application, a location of the second version of the binary.

10. The method of claim **8**, wherein the application is an in-memory database.

11. The method of claim **10**, wherein the binary is an executable to perform backups of the in-memory database.

12. The method of claim **10**, wherein the binary is a driver to connect the in-memory database to a data lake.

13. The method of claim **8**, wherein the container-based virtualization system is Docker.

14. The method of claim **8**, wherein the container-orchestration system is Kubernetes.

15. A non-transitory machine-readable medium storing instructions which, when executed by one or more processors, cause the one or more processors to perform operations comprising:

creating a first pod in a container-orchestration system, the first pod containing a reference to an image including an application;

running the first pod on a first host, the running including:

retrieving the image including the application;

launching a first container in a container-based virtualization system;

starting the application; and

mounting a shared volume host path to the first pod;

creating a second pod in a container-orchestration system, the second pod containing a reference to an image including a first version of the binary associated with the application, wherein the application contains one or more calls that, when executed, cause the binary to perform one or more operations;

running the second pod on the first host, the running including:

retrieving the image including the first version of the binary;

launching a second container in the container-based virtualization system; and

mounting the shared volume host path to the second pod, the shared volume host path identifying, to the application, a location of the first version of the binary.

16. The non-transitory machine-readable medium of claim **15**, wherein the operations further comprise:

creating a third pod in a container-orchestration system, the third pod containing a reference to an image including a second version of the binary associated with the application;

killing the second pod;

running the third pod on the first host, the running including:

retrieving the image including the second version of the binary;

launching a third container in the container-based virtualization system;

mounting the shared volume host path to the third pod; and

updating the shared volume host path to identify, to the application, a location of the second version of the binary.

17. The non-transitory machine-readable medium of claim **15**, wherein the application is an in-memory database.

18. The non-transitory machine-readable medium of claim **17**, wherein the binary is an executable to perform backups of the in-memory database.

19. The non-transitory machine-readable medium of claim **17**, wherein the binary is a driver to connect the in-memory database to a data lake.

20. The non-transitory machine-readable medium of claim **15**, wherein the container-based virtualization system is Docker.

* * * * *